第5章

Pandas数据载入与预处理

对于数据分析而言，数据大部分来源于外部数据，如常用的 CSV 文件、Excel 文件和数据库文件等。Pandas 库将外部数据转换为 DataFrame 数据格式，处理完成后再存储到相应的外部文件中。


视频讲解

5.1 数据载入

5.1.1 读/写文本文件

1. 文本文件读取

文本文件是一种由若干行字符构成的计算机文件，它是一种典型的顺序文件。CSV 是一种逗号分隔的文件格式，因为其分隔符不一定是逗号，又被称为字符分隔文件，文件以纯文本形式存储表格数据(数字和文本)。

在 Pandas 中使用 read_table 函数来读取文本文件：

```
pandas.read_table(filepath_or_buffer, sep = '\t', header = 'infer', names = None, index_col = None, dtype = None, engine = None, nrows = None)
```

在 Pandas 中使用 read_csv 函数来读取 CSV 文件：

```
pandas.read_csv(filepath_or_buffer, sep = ',', header = 'infer', names = None, index_col = None, dtype = None, engine = None, nrows = None)
```

两个文本文件读取方法中的常用参数及其说明见表 5-1。

表 5-1 read_table 和 read_csv 常用参数及其说明

参　数	说　明
filepath	接收 string,代表文件路径,无默认
sep	接收 string,代表分隔符。read_csv 默认为“,”,read_table 默认为制表符“[Tab]”,如果分隔符指定错误,在读取数据的时候,每一行数据将连成一片
header	接收 int 或 sequence,表示将某行数据作为列名,默认为 infer,表示自动识别
names	接收 array,表示列名,默认为 None
index_col	接收 int、sequence 或 False,表示索引列的位置,取值为 sequence 则代表多重索引,默认为 None
dtype	接收 dict,代表写入的数据类型(列名为 key,数据格式为 values),默认为 None
engine	接收 c 或者 python,代表数据解析引擎,默认为 c
nrows	接收 int,表示读取前 n 行,默认为 None

【例 5-1】 使用 read_csv 函数读取 CSV 文件。

```
In[1]:  df1 = pd. read_csv('文件路径文件名')
        #读取 CSV 文件到 DataFrame 中
        df2 = pd. read_table('文件路径文件名', sep = ',')
        #使用 read_table,并指定分隔符
        df3 = pd. read_csv('文件路径文件名',names = ['a','b', --- ])
        #文件不包含表头行,允许自动分配默认列名,也可以指定列名
```

2. 文本文件的存储

文本文件的存储和读取类似,结构化数据可以通过 Pandas 中的 to_csv 函数实现以 CSV 文件格式存储文件。

```
DataFrame.to_csv(path_or_buf = None, sep = ',', na_rep = '', columns = None, header = True,
index = True, index_label = None, mode = 'w', encoding = None)
```

5.1.2 读/写 Excel 文件

1. Excel 文件的读取

Pandas 提供了 read_excel 函数读取 xls 和 xlsx 两种 Excel 文件,其格式为:

```
pandas.read_excel(io, sheetname, header = 0, index_col = None, names = None, dtype)
```

read_excel 函数和 read_table 函数的部分参数相同,其常用参数及其说明见表 5-2。

表 5-2 Pandas 读/写 Excel 文件

参　数	说　明
io	接收 string,表示文件路径,无默认
sheetname	接收 string、int,代表 Excel 表内数据的分表位置,默认为 0

续表

参　数	说　明
header	接收 int 或 sequence，表示将某行数据作为列名，默认为 infer，表示自动识别
names	接收 int、sequence 或者 False，表示索引列的位置，取值为 sequence 则代表多重索引，默认为 None
index_col	接收 int、sequence 或者 False，表示索引列的位置，取值为 sequence 则代表多重索引，默认为 None
dtype	接收 dict，代表写入的数据类型(列名为 key，数据格式为 values)，默认为 None

【例 5-2】 读取 Excel 文件。

```
In[2]:  xlsx = pd.excelFile('example/ex1.xlsx')
        pd.read_excel(xlsx, 'Sheet1')
        #也可以直接使用
        frame = pd.read_excel('example/ex1.xlsx', 'Sheet1')
```

2. Excel 文件的存储

将文件存储为 Excel 文件，可以使用 to_excel 方法。其语法格式如下：

```
DataFrame.to_excel(excel_writer = None, sheetname = None', na_rep = '', header = True, index = True, index_label = None, mode = 'w', encoding = None)
```

to_excel 与 to_csv 方法的常用参数基本一致，区别之处在于 to_excel 指定存储文件的文件路径参数名称为 excel_writer，并且没有 sep 参数，增加了一个 sheetnames 参数用来指定存储的 Excel Sheet 的名称，默认为 sheet1。

5.1.3 JSON 数据的读取与存储

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式，其简洁和清晰的层次结构使其成为理想的数据交换语言。JSON 数据使用大括号来区分表示并存储。

1. Pandas 读取 JSON 数据

Pandas 通过 read_json 函数读取 JSON 数据，读取时会出现顺序错乱的问题，因此要对行索引进行排序。读取代码如下：

```
import pandas as pd
df = pd.read_json('FileName')
df = df.sort_index
```

2. JSON 数据的存储

Pandas 使用 pd.to_json 实现将 DataFrame 数据存储为 JSON 文件。

5.1.4 读取数据库文件

在许多应用中,很多文件来源于数据库。Pandas 实现了便捷连接并存取数据库文件的方法。

1. Pandas 读取 MySQL 数据

要读取 MySQL 中的数据,首先要安装 MySQLdb 包,然后进行数据文件读取。读取代码如下:

```
import pandas as pd
import MySQLdb
conn = MySQLdb.connect(host = host,port = port,user = username,passwd = password,db =
db_name)
df = pd.read_sql('select * from table_name',con = conn)
conn.close()
```

2. Pandas 读取 SQL Server 中的数据

要读取 SQL Server 中的数据,首先要安装 pymssql 包,然后进行数据文件读取。读取代码如下:

```
import pandas as pd
import pymssql
conn = pymssql.connect(host = host, port = port , user = username, password = password,
database = database)
df = pd.read_sql("select * from table_name",con = conn)
conn.close()
```

5.2 合并数据

在实际的数据分析中,可能有不同的数据来源,因此需要对数据进行合并处理。

5.2.1 merge 数据合并

Python 中的 merge 函数是通过一个或多个键将两个 DataFrame 按行合并起来,与 SQL 中的 join 用法类似,Pandas 中的数据合并函数 merge()格式如下:

```
merge(left, right, how = 'inner', on = None, left_on = None, right_on = None, left_index =
False, right_index = False, sort = False, suffixes = ('_x', '_y'), copy = True, indicator =
False, validate = None)
```

merge 方法的主要参数及其说明见表 5-3。

表 5-3 merge 方法的主要参数及其说明

参 数	说 明
left	参与合并的左侧 DataFrame
right	参与合并的右侧 DataFrame
how	连接方法：inner,left,right,outer
on	用于连接的列名
left_on	左侧 DataFrame 中用于连接键的列
right_on	右侧 DataFrame 中用于连接键的列
left_index	左侧 DataFrame 中行索引作为连接键
right_index	右侧 DataFrame 中行索引作为连接键
sort	合并后会对数据排序,默认为 True
suffixes	修改重复名

【例 5-3】 merge 的默认合并数据。

```
In[3]:  price = pd.DataFrame({'fruit':['apple','grape','orange','orange'],
        'price':[8,7,9,11]})
        amount = pd.DataFrame({'fruit':['apple','grape','orange'],'amout':[5,11,8]})
        display(price,amount,pd.merge(price,amount))
```

Out[3]:

	fruit	price
0	apple	8
1	grape	7
2	orange	9
3	orange	11

	fruit	amout
0	apple	5
1	grape	11
2	orange	8

	fruit	price	amout
0	apple	8	5
1	grape	7	11
2	orange	9	8
3	orange	11	8

由于两个 DataFrame 都有 fruit 列,所以默认按照该列进行合并,默认 how＝'inner',即 pd.merge(amount,price,on＝'fruit',how＝'inner')。如果两个 DataFrame 的列名不相同,可以单独指定。

【例 5-4】 指定合并时的列名。

```
In[4]:  display(pd.merge(price,amount,left_on = 'fruit',right_on = 'fruit'))
```

Out[4]:

	fruit	price	amout
0	apple	8	5
1	grape	7	11
2	orange	9	8
3	orange	11	8

merge 合并时默认是内连接(inner),即返回交集。通过 how 参数可以选择连接方法:左连接(left)、右连接(right)和外连接(outer)。

【例 5-5】 左连接。

```
In[5]:  display(pd.merge(price,amount,how = 'left'))
```

Out[5]:

	fruit	price	amout
0	apple	8	5
1	grape	7	11
2	orange	9	8
3	orange	11	8

【例 5-6】 右连接。

```
In[6]:  display(pd.merge(price,amount,how = 'right'))
```

Out[6]:

	fruit	price	amout
0	apple	8	5
1	grape	7	11
2	orange	9	8
3	orange	11	8

也可以通过多个键进行合并。

【例 5-7】 merge 通过多个键合并。

```
In[7]:  left = pd.DataFrame({'key1':['one','one','two'],'key2':['a','b','a'],
        'value1':range(3)})
        right = pd.DataFrame({'key1':['one','one','two','two'],'key2':['a','a','a','b'],
        'value2':range(4)})
        display(left,right,pd.merge(left,right,on = ['key1','key2'],how = 'left'))
```

Out[7]:

	key1	key2	value1
0	one	a	0
1	one	b	1
2	two	a	2

	key1	key2	value2
0	one	a	0
1	one	a	1
2	two	a	2
3	two	b	3

	key1	key2	value1	value2
0	one	a	0	0.0
1	one	a	0	1.0
2	one	b	1	NaN
3	two	a	2	2.0

在合并时会出现重复列名，虽然可以人为进行重复列名的修改，但 merge 函数提供了 suffixes 用于处理该问题。

【例 5-8】 merge 函数中参数 suffixes 的应用。

```
In[8]:  print(pd.merge(left,right,on = 'key1'))
        print(pd.merge(left,right,on = 'key1',suffixes = ('_left','_right')))
Out[8]: key1 key2_x    value1 key2_y    value2
        0    one         a    0         a    0
        1    one         a    0         a    1
        2    one         b    1         a    0
        3    one         b    1         a    1
        4    two         a    2         a    2
        5    two         a    2         b    3
        key1 key2_left  value1 key2_right  value2
        0    one         a    0         a    0
        1    one         a    0         a    1
        2    one         b    1         a    0
        3    one         b    1         a    1
        4    two         a    2         a    2
        5    two         a    2         b    3
```

5.2.2 concat 数据连接

如果要合并的 DataFrame 之间没有连接键，就无法使用 merge 方法，可以使用 Pandas 中的 concat 方法。默认情况下会按行的方向堆叠数据；如果在列向上连接，设置 axis＝1 即可。

【例 5-9】 两个 Series 的数据连接。

```
In[9]:  s1 = pd.Series([0,1],index = ['a','b'])
        s2 = pd.Series([2,3,4],index = ['a','d','e'])
        s3 = pd.Series([5,6],index = ['f','g'])
        print(pd.concat([s1,s2,s3]))                    #Series 行合并
Out[9]: a    0
        b    1
        a    2
        d    3
        e    4
        f    5
        g    6
        dtype: int64
```

【例 5-10】 两个 DataFrame 的数据连接。

```
In[10]:  data1 = pd.DataFrame(np.arange(6).reshape(2,3),columns = list('abc'))
         data2 = pd.DataFrame(np.arange(20,26).reshape(2,3),columns = list('ayz'))
         data = pd.concat([data1,data2],axis = 0)
         display(data1,data2,data)
```

Out[10]:

	a	b	c
0	0	1	2
1	3	4	5

	a	y	z
0	20	21	22
1	23	24	25

	a	b	c	y	z
0	0	1.0	2.0	NaN	NaN
1	3	4.0	5.0	NaN	NaN
0	20	NaN	NaN	21.0	22.0
1	23	NaN	NaN	24.0	25.0

通过结果可以看出,concat 连接方式为外连接(并集),通过传入 join='inner'可以实现内连接。

可以通过 join_axis 指定使用的索引顺序。

【例 5-11】 指定索引顺序。

```
In[11]:  s1 = pd.Series([0,1],index = ['a','b'])
         s2 = pd.Series([2,3,4],index = ['a','d','e'])
         s3 = pd.Series([5,6],index = ['f','g'])
         s4 = pd.concat([s1 * 5,s3],sort = False)
         s5 = pd.concat([s1,s4],axis = 1,sort = False)
         s6 = pd.concat([s1,s4],axis = 1,join = 'inner',sort = False)
         s7 = pd.concat([s1,s4],axis = 1,join = 'inner',join_axis = [['b','a']],
         sort = False)
         display(s4,s5,s6,s7)
```

Out[11]:

```
a    0
b    5
f    5
g    6
dtype: int64
```

	0	1
a	0.0	0
b	1.0	5
f	NaN	5
g	NaN	6

	0	1
a	0	0
b	1	5

	0	1
b	1	5
a	0	0

5.2.3 combine_first 合并数据

如果需要合并的两个 DataFrame 存在重复索引，则使用 merge 和 concat 都无法正确合并，此时需要使用 combine_first 方法。数据 w1 和 w2 分别如下所示：

w1

	0	1
a	0	0
b	1	5

w2

	0	1
a	0.0	0
b	1.0	5
f	NaN	5
g	NaN	6

【例 5-12】 使用 combine_first 合并。

```
In[12]:   w1.combine_first(w2)
```

Out[12]:

	0	1
a	0.0	0.0
b	1.0	5.0
f	NaN	5.0
g	NaN	6.0


视频讲解

5.3 数据清洗

数据一般是不完整、有噪声和不一致的。数据清洗试图填充缺失的数据值、光滑噪声、识别离群点并纠正数据中的不一致。

5.3.1 检测与处理缺失值

在许多数据分析工作中，经常会有缺失数据的情况。Pandas 的目标之一就是尽量轻松地处理缺失数据。

1. 缺失值的处理

Pandas 对象的所有描述性统计默认都不包括缺失数据。对于数值数据，Pandas 使用浮点值 NaN 表示缺失数据。

1) 缺失值的检测与统计

函数 isnull()可以直接判断该列中的哪个数据为 NaN。

【例 5-13】 使用 isnull 检测缺失值。

```
In[13]:   string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
          print(string_data)
          string_data.isnull()
Out[13]:  0     aardvark
          1    artichoke
```

```
2          NaN
3      avocado
dtype: object
0    False
1    False
2     True
3    False
dtype: bool
```

在 Pandas 中,缺失值表示为 NA,它表示不可用(not available)。在统计应用中,NA 数据可能是不存在的数据,或者存在却没有观察到的数据(例如数据采集中发生了问题)。当清洗数据用于分析时,最好直接对缺失数据进行分析,以判断数据采集问题或缺失数据可能导致的偏差。Python 内置的 None 值也会被当作 NA 处理。

【例 5-14】 Series 中的 None 值处理。

```
In[14]:  string_data = pd.Series(['aardvark', 'artichoke',np.nan, 'avocado'])
         string_data.isnull()
Out[14]: 0    False
         1    False
         2     True
         3    False
         dtype: bool
```

2) 缺失值的统计

【例 5-15】 使用 isnull().sum()统计缺失值。

```
In[15]:  df = pd.DataFrame(np.arange(12).reshape(3,4),columns = ['A','B','C','D'])
         df.ix[2,:] = np.nan
         df[3] = np.nan
         print(df)
         df.isnull().sum()
Out[15]:       A    B    C    D    3
         0   0.0  1.0  2.0  3.0  NaN
         1   4.0  5.0  6.0  7.0  NaN
         2   NaN  NaN  NaN  NaN  NaN
         A     1
         B     1
         C     1
         D     1
         3     3
         dtype: int64
```

另外,通过 info 方法,也可以查看 DataFrame 每列数据的缺失情况。

【例 5-16】 用 info 方法查看 DataFrame 的缺失值。

```
In[16]:  df.info()
Out[16]: <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 3 entries, 0 to 2
         Data columns (total 5 columns):
```

```
A    2 non-null float64
B    2 non-null float64
C    2 non-null float64
D    2 non-null float64
3    0 non-null float64
dtypes: float64(5)
memory usage: 200.0 bytes
```

2. 缺失值的处理

1）删除缺失值

在缺失值的处理方法中，删除缺失值是常用的方法之一。通过 dropna 方法可以删除具有缺失值的行。

dropna 方法的格式：

```
dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
```

dropna 的参数及其说明见表 5-4。

表 5-4 dropna 的参数及其说明

参 数	说 明
axis	默认为 axis=0，当某行出现缺失值时，将该行丢弃并返回；当 axis=1，且某列出现缺失值时，将该列丢弃
how	确定缺失值个数，缺省时 how='any'，how='any'表明只要某行有缺失值就将该行丢弃，how='all'表明某行全部为缺失值才将其丢弃
thresh	阈值设定，当行列中非缺失值的数量少于给定的值就将该行丢弃
subset	部分标签中删除某行列，例如 subset=['a','d']，即丢弃子列 a d 中含有缺失值的行
inplace	bool 取值，默认为 False，当 inplace= True，即对原数据操作，无返回值

对于 Series，dropna 返回一个仅含非空数据和索引值的 Series。

【例 5-17】 Series 的 dropna 用法。

```
In[17]:  from numpy import nan as NA
         data = pd.Series([1, NA, 3.5, NA, 7])
         print(data)
         print(data.dropna())
Out[17]: 0    1.0
         1    NaN
         2    3.5
         3    NaN
         4    7.0
         dtype: float64
         0    1.0
         2    3.5
         4    7.0
         dtype: float64
```

当然,也可以通过布尔型索引达到这个目的。

【例 5-18】 布尔型索引选择过滤非缺失值。

```
In[18]:  not_null = data.notnull()
         print(not_null)
         print(data[not_null])
Out[18]: 0    True
         1   False
         2    True
         3   False
         4    True
         dtype: bool
         0    1.0
         2    3.5
         4    7.0
         dtype: float64
```

对于 DataFrame 对象,dropna 默认丢弃任何含有缺失值的行。

【例 5-19】 DataFrame 对象的 dropna 默认参数使用。

```
In[19]:  from numpy import nan as NA
         data = pd.DataFrame([[1., 5.5, 3.], [1., NA, NA],[NA, NA, NA],
         [NA, 5.5, 3.]])
         print(data)
         cleaned = data.dropna()
         print('删除缺失值后的:\n',cleaned)
Out[19]:      0    1    2
         0   1.0  5.5  3.0
         1   1.0  NaN  NaN
         2   NaN  NaN  NaN
         3   NaN  5.5  3.0
         删除缺失值后的:
              0    1    2
         0    1.0  5.5  3.0
```

传入 how='all'将只丢弃全为 NA 的那些行。

【例 5-20】 传入参数 all。

```
In[20]:  data = pd.DataFrame([[1., 5.5, 3.], [1., NA, NA],[NA, NA, NA],
         [NA, 5.5, 3.]])
         print(data)
         data.dropna(how = 'all')
Out[20]:      0    1    2
         0  1.0  5.5  3.0
         1  1.0  NaN  NaN
         2  NaN  NaN  NaN
         3  NaN  5.5  3.0
```

	0	1	2
0	1.0	5.5	3.0
1	1.0	NaN	NaN
3	NaN	5.5	3.0

如果用同样的方式丢弃 DataFrame 的列，只需传入 axis = 1 即可。

【例 5-21】 dropna 中的 axis 参数应用。

```
In[21]:  data = pd.DataFrame([[1., 5.5, NA], [1., NA, NA],[NA, NA, NA], [NA, 5.5, NA]])
         print(data)
         data.dropna(axis = 1, how = 'all')
```

```
Out[21]:      0    1   2
          0  1.0  5.5 NaN
          1  1.0  NaN NaN
          2  NaN  NaN NaN
          3  NaN  5.5 NaN
```

	0	1
0	1.0	5.5
1	1.0	NaN
2	NaN	NaN
3	NaN	5.5

使用 thresh 参数，当传入 thresh=N 时，表示要求一行至少具有 N 个非 NaN 才能存活。例如 df.dropna(thresh=len(df) * 0.9, axis=1) 表示如果某列数据中的缺失值超过 10%，则删除该列。

【例 5-22】 dropna 中的 thresh 参数应用。

```
In[22]:  df = pd.DataFrame(np.random.randn(7, 3))
         df.iloc[:4, 1] = NA
         df.iloc[:2, 2] = NA
         print(df)
         df.dropna(thresh = 2)
```

```
Out[22]:          0         1         2
          0  0.176209       NaN       NaN
          1 -0.871199       NaN       NaN
          2  1.624651       NaN  0.829676
          3 -0.286038       NaN -1.809713
          4 -0.640662  0.666998 -0.032702
          5 -0.453412 -0.708945  1.043190
          6 -0.040305 -0.290658 -0.089056
```

	0	1	2
2	1.624651	NaN	0.829676
3	-0.286038	NaN	-1.809713
4	-0.640662	0.666998	-0.032702
5	-0.453412	-0.708945	1.043190
6	-0.040305	-0.290658	-0.089056

2）填充缺失值

直接删除有缺失值的样本并不是一个很好的方法，因此可以用一个特定的值替换缺失值。缺失值所在的特征为数值型时，通常使用其均值、中位数和众数等描述其集中趋势的统计量来填充；缺失值所在特征为类别型数据时，则选择众数来填充。Pandas 库中

提供了缺失值替换的方法 fillna。

fillna 的格式：

```
pandas.DataFrame.fillna(value = None,method = None,axis = None,inplace = False,
                        limit = None)
```

fillna 参数及其说明见表 5-5。

表 5-5　fillna 参数及其说明

参　数	说　明
value	用于填充缺失值的标量值或字典对象
method	插值方式
axis	待填充的轴，默认为 axis=0
inplace	修改调用者对象而不产生副本
limit	(对于前向和后向填充)可以连续填充的最大数量

通过一个常数调用 fillna 就会将缺失值替换为那个常数值，例如 df.fillna(0)用零代替缺失值；也可以通过一个字典调用 fillna，就可以实现对不同的列填充不同的值。

【例 5-23】 通过字典形式填充缺失值。

```
In[23]:   df = pd.DataFrame(np.random.randn(5,3))
          df.loc[:3,1] = NA
          df.loc[:2,2] = NA
          print(df)
          df.fillna({1:0.88,2:0.66})
```

```
Out[23]:          0         1         2
          0  0.861692       NaN       NaN
          1  0.911292       NaN       NaN
          2  0.465258       NaN       NaN
          3 -0.797297       NaN -0.342404
          4  0.658408  0.872754 -0.108814
```

	0	1	2
0	0.861692	0.880000	0.660000
1	0.911292	0.880000	0.660000
2	0.465258	0.880000	0.660000
3	-0.797297	0.880000	-0.342404
4	0.658408	0.872754	-0.108814

fillna 默认会返回新对象，但也可以通过参数 inplace=True 对现有对象进行就地修改。对 reindex 有效的那些插值方法也可用于 fillna。

【例 5-24】 fillna 中 method 的应用。

```
In[24]:   df = pd.DataFrame(np.random.randn(6, 3))
          df.iloc[2:, 1] = NA
          df.iloc[4:, 2] = NA
          print(df)
```

```
In[24]: df.fillna(method = 'ffill')
Out[24]:
          0         1         2
0 -1.180338 -0.663622  0.952264
1 -0.219780 -1.356420  0.742720
2 -2.169303       NaN  1.129426
3  0.139349       NaN -1.463485
4  1.327619       NaN       NaN
5  0.834232       NaN       NaN
```

	0	1	2
0	-1.180338	-0.663622	0.952264
1	-0.219780	-1.356420	0.742720
2	-2.169303	-1.356420	1.129426
3	0.139349	-1.356420	-1.463485
4	1.327619	-1.356420	-1.463485
5	0.834232	-1.356420	-1.463485

可以使用 fillna 实现许多别的功能，例如可以传入 Series 的平均值或中位数。

【例 5-25】 用 Series 的均值填充。

```
In[25]: data = pd.Series([1., NA, 3.5, NA, 7])
        data.fillna(data.mean())
Out[25]: 0    1.000000
         1    3.833333
         2    3.500000
         3    3.833333
         4    7.000000
         dtype: float64
```

【例 5-26】 在 DataFrame 中用均值填充。

```
In[26]: df = pd.DataFrame(np.random.randn(4, 3))
        df.iloc[2:, 1] = NA
        df.iloc[3:, 2] = NA
        print(df)
        df[1] = df[1].fillna(df[1].mean())
        print(df)
Out[26]:           0         1           2
        0   0.656155  0.008442    0.025324
        1   0.160845  0.829127    1.065358
        2  -0.321155       NaN   -0.955008
        3   0.953510       NaN         NaN
                   0         1           2
        0   0.656155  0.008442    0.025324
        1   0.160845  0.829127    1.065358
        2  -0.321155  0.418785   -0.955008
        3   0.953510  0.418785         NaN
```

对于 fillna 的参数,可以通过“df. fillna?”进行帮助查看。

5.3.2 检测与处理重复值

数据中存在重复样本时只需保留一份即可,其余的可以做删除处理。在 DataFrame 中使用 duplicates 方法判断各行是否有重复数据。duplicates 方法返回一个布尔值的 Series,反映每一行是否与之前的行重复。

【例 5-27】 判断 DataFrame 中的重复数据。

```
In[27]:  data = pd.DataFrame({ 'k1':['one','two'] * 3 + ['two'],'k2':[1, 1, 2, 3, 1, 4, 4],
         'k3':[1,1,5,2,1, 4, 4] })
         print(data)
         data.duplicated()
Out[27]:      k1      k2      k3
         0    one      1       1
         1    two      1       1
         2    one      2       5
         3    two      3       2
         4    one      1       1
         5    two      4       4
         6    two      4       4
         0     False
         1     False
         2     False
         3     False
         4      True
         5     False
         6      True
         dtype: bool
```

Pandas 通过 drop_duplicates 删除重复的行,drop_duplicates 方法的格式为:

```
pandas.DataFrame(Series). drop_duplicates(self, subset = None, keep = 'first', inplace =
False)
```

该方法常用的参数及其说明见表 5-6。

表 5-6 drop_duplicates 的主要参数及其说明

参 数	说 明
subset	接收 string 或 sequence,表示进行去重的列,默认全部列
keep	接收特定 string,表示重复时保留第几个数据,'first'保留第一个,'last'保留最后一个,'False'只要有重复都不保留,默认为 first
inplace	接收布尔型数据,表示是否在原表上进行操作,默认为 False

使用 drop_duplicates 方法去重时,当且仅当 subset 参数中的特征重复时才会执行去重操作,去重时可以选择保留哪一个或者不保留。

【例 5-28】 每行各个字段都相同时去重。

```
In[28]:  data.drop_duplicates()
Out[28]:
```

	k1	k2	k3
0	one	1	1
1	two	1	1
2	one	2	5
3	two	3	2
5	two	4	4

【例 5-29】 指定部分列重复时去重。

```
In[29]:  data.drop_duplicates(['k2','k3'])
Out[29]:
```

	k1	k2	k3
0	one	1	1
2	one	2	5
3	two	3	2
5	two	4	4

默认保留的数据为第一个出现的记录，通过传入 keep='last'可以保留最后一个出现的记录。

【例 5-30】 去重时保留最后出现的记录。

```
In[30]:  data.drop_duplicates(['k2','k3'],keep = 'last')
Out[30]:
```

	k1	k2	k3
2	one	2	5
3	two	3	2
4	one	1	1
6	two	4	4

5.3.3 检测与处理异常值

异常值是指数据中存在的个别数值明显偏离其余数据的值。异常值的存在会严重干扰数据分析的结果，因此经常要检验数据中是否有输入错误或含有不合理的数据。在数据统计方法中一般常用散点图、箱线图和 3σ 法则检测异常值。

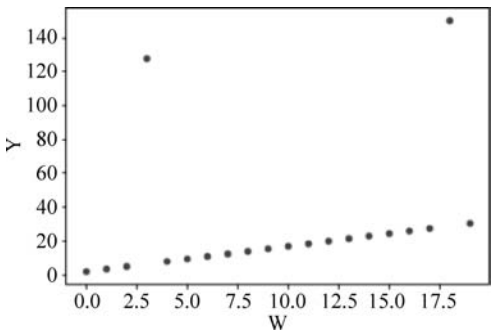
1. 散点图方法

通过数据分布的散点图发现异常数据。

【例 5-31】 使用散点图检测异常值。

```
In[31]:  wdf = pd.DataFrame(np.arange(20),columns = ['W'])
         wdf['Y'] = wdf['W'] * 1.5 + 2
         wdf.iloc[3,1] = 128
         wdf.iloc[18,1] = 150
```

```
In[31]:  wdf.plot(kind = 'scatter',x = 'W',y = 'Y')
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x2680853ca20>
```

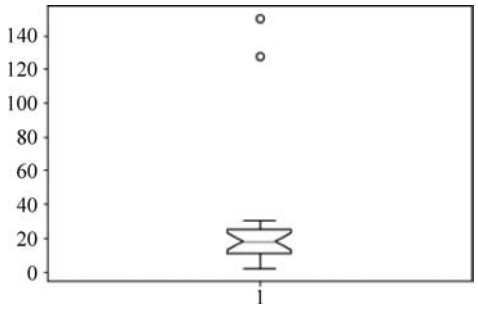


2. 箱线图分析

箱线图使用数据中的 5 个统计量(最小值、下四分位数 Q_1、中位数 Q_2、上四分位数 Q_3 和最大值)来描述数据,它也可以粗略地看出数据是否具有对称性、分布的分散程度等信息。利用箱线图还可以进行异常值检测。在检测过程中,根据经验,将最大(最小)值设置为与四分位数值间距为 1.5 个 IQR(IQR=Q_3-Q_2)的值,即 min=$Q_1-1.5$IQR,max=$Q_3+1.5$IQR,小于 min 和大于 max 的值被认为是异常值。

【例 5-32】 使用箱线图分析异常值。

```
In[32]:  import matplotlib.pyplot as plt
         plt.boxplot(wdf['Y'].values,notch = True)
Out[32]:
```



3. 3σ 法则

若数据服从正态分布,在 3σ 原则下,异常值被定义为一组测定值中与平均值的偏差超过 3 倍标准差的值,因为在正态分布的假设下,距离平均值 3σ 之外的值出现的概率小于 0.003。因此根据小概率事件,可以认为超出 3σ 之外的值为异常数据。

【例 5-33】 使用 3σ 法则检测异常值。

```
In[33]:  def outRange(S):
             blidx = (S.mean() - 3 * S.std()> S)|(S.mean() + 3 * S.std()< S)
             idx = np.arange(S.shape[0])[blidx]
             outRange = S.iloc[idx]
```

```
            return outRange
        outier = outRange(wdf['Y'])
        outier
Out[33]: 18    150.0
Name: Y, dtype: float64
```

5.3.4 数据转换

1. 数据值替换

数据值替换是将查询到的数据替换为指定数据。在 Pandas 中通过 replace 进行数据值的替换。

【例 5-34】 使用 replace 替换数据值。

```
In[34]:  data = {'姓名':['李红','小明','马芳','国志'],'性别':['0','1','0','1'],
         '籍贯':['北京','甘肃','','上海']}
         df = pd.DataFrame(data)
         df = df.replace('','不详')
         print(df)
Out[34]:    姓名  性别   籍贯
         0  李红    0   北京
         1  小明    1   甘肃
         2  马芳    0   不详
         3  国志    1   上海
```

也可以同时对不同值进行多值替换，参数传入的方式可以是列表，也可以是字典格式。传入的列表中，第一个列表为被替换的值，第二个列表是对应替换的值。

【例 5-35】 使用 replace 传入列表实现多值替换。

```
In[35]:  df = df.replace(['不详','甘肃'],['兰州','兰州'])
         print(df)
Out[35]:    姓名  性别   籍贯
         0  李红    0   北京
         1  小明    1   兰州
         2  马芳    0   兰州
         3  国志    1   上海
```

【例 5-36】 使用 replace 传入字典实现多值替换。

```
In[36]:  df = df.replace({'1':'男','0':'女'})
         print(df)
Out[36]:    姓名  性别   籍贯
         0  李红   女   北京
         1  小明   男   兰州
         2  马芳   女   兰州
         3  国志   男   上海
```

2. 使用函数或映射进行数据转换

在数据分析中,经常需要进行数据的映射或转换,在 Pandas 中可以自定义函数,然后通过 map 方法实现。

【例 5-37】 使用 map 方法映射数据。

```
In[37]:  data = {'姓名':['李红','小明','马芳','国志'],'性别':['0','1','0','1'],
         '籍贯':['北京','兰州','兰州','上海']}
         df = pd.DataFrame(data)
         df['成绩'] = [58,86,91,78]
         print(df)
         def grade(x):
             if x>=90:
                 return '优'
             elif 70<=x<90:
                 return '良'
             elif 60<=x<70:
                 return '中'
             else:
                 return '差'
         df['等级'] = df['成绩'].map(grade)
         print(df)
Out[37]:      姓名  性别    籍贯  成绩
         0    李红     0    北京    58
         1    小明     1    兰州    86
         2    马芳     0    兰州    91
         3    国志     1    上海    78
              姓名  性别    籍贯  成绩    等级
         0    李红     0    北京    58     差
         1    小明     1    兰州    86     良
         2    马芳     0    兰州    91     优
         3    国志     1    上海    78     良
```

5.4 数据标准化

不同特征之间往往具有不同的量纲,由此造成数值之间的差异。为了消除特征之间量纲和取值范围的差异可能会造成的影响,需要对数据进行标准化处理。

5.4.1 离差标准化数据

离差标准化是对原始数据所做的一种线性变换,将原始数据的数值映射到[0,1]。转换公式如下:

$$x_1 = \frac{x - \min}{\max - \min} \tag{5.1}$$

【例 5-38】 数据的离差标准化。

```
In[38]:  def MinMaxScale(data):
             data = (data - data.min())/(data.max() - data.min())
             return data
         x = np.array([[ 1., -1., 2.],[ 2., 0., 0.],[ 0., 1., -1.]])
         print('原始数据为:\n',x)
         x_scaled = MinMaxScale(x)
         print('标准化后矩阵为:\n',x_scaled,end = '\n')
Out[38]: 原始数据为:
         [[  1.  -1.   2.]
         [   2.   0.   0.]
         [   0.   1.  -1.]]
         标准化后矩阵为:
         [[0.66666667  0.          1.        ]
         [ 1.          0.33333333  0.33333333]
         [ 0.33333333  0.66666667  0.        ]]
```

5.4.2 标准差标准化数据

标准差标准化又称为零均值标准化或 z 分数标准化，是当前使用最广泛的数据标准化方法。经过该方法处理的数据均值为 0，标准差为 1，转换公式如下：

$$x_1 = \frac{x - mean}{std} \tag{5.2}$$

【例 5-39】 数据的标准差标准化。

```
In[39]:  def StandardScale(data):
             data = (data - data.mean())/data.std()
             return data
         x = np.array([[ 1., -1., 2.],[ 2., 0., 0.],[ 0., 1., -1.]])
         print('原始数据为:\n',x)
         x_scaled = StandardScale(x)
         print('标准化后矩阵为:\n',x_scaled,end = '\n')
Out[39]: 原始数据为:
         [[  1. -1.   2. ]
         [   2.  0.   0. ]
         [   0.  1. -1.]]
         标准化后矩阵为:
         [[  0.52128604  -1.35534369   1.4596009  ]
         [   1.4596009   -0.41702883  -0.41702883 ]
         [  -0.41702883   0.52128604  -1.35534369]]
```


视频讲解

5.5 数据变换与数据离散化

数据分析的预处理除了数据清洗、数据合并和标准化之外，还包括数据变换的过程，如类别型数据变换和连续型数据的离散化。

5.5.1 类别型数据的哑变量处理

类别型数据是数据分析中十分常见的特征变量,但是在进行建模时,Python 不能像 R 那样去直接处理非数值型的变量,因此往往需要对这些类别变量进行一系列转换,如哑变量。

哑变量(Dummy Variables)又称为虚拟变量,是用以反映质的属性的一个人工变量,是量化了的质变量,通常取值为 0 或 1。Python 中使用 Pandas 库中的 get_dummies 函数对类别型特征进行哑变量处理。

【例 5-40】 数据的哑变量处理。

```
In[40]:  df = pd.DataFrame([
                    ['green', 'M', 10.1, 'class1'],
                    ['red', 'L', 13.5, 'class2'],
                    ['blue', 'XL', 15.3, 'class1']])
         df.columns = ['color', 'size', 'prize','class label']
         print(df)
         pd.get_dummies(df)
Out[40]:    color  size  prize  class  label
         0  green     M   10.1         class1
         1    red     L   13.5         class2
         2   blue    XL   15.3         class1
```

	prize	color_blue	color_green	color_red	size_L	size_M	size_XL	class label_class1	class label_class2
0	10.1	0	1	0	0	1	0	1	0
1	13.5	0	0	1	1	0	0	0	1
2	15.3	1	0	0	0	0	1	1	0

对于一个类别型特征,若取值有 m 个,则经过哑变量处理后就变成了 m 个二元互斥特征,每次只有一个激活,使得数据变得稀疏。

5.5.2 连续型变量的离散化

数据分析和统计的预处理阶段,经常会碰到年龄、消费等连续型数值,而很多模型算法,尤其是分类算法,都要求数据是离散的,因此要将数值进行离散化分段统计以提高数据区分度。

常用的离散化方法主要有等宽法、等频法和聚类分析法。

1. 等宽法

将数据的值域划分成具有相同宽度的区间,区间个数由数据本身的特点决定或由用户指定。Pandas 提供了 cut 函数,可以进行连续型数据的等宽离散化。cut 函数的基本语法格式为:

```
pandas.cut(x,bins,right = True,labels = None,retbins = False,precision = 3)
```

cut 函数的主要参数及其说明见表 5-7。

表 5-7 cut 函数的主要参数及其说明

参 数	说 明
x	接收 array 或 Series,待离散化的数据
bins	接收 int、list、array 和 tuple。若为 int 指离散化后的类别数目,若为序列型则表示进行切分的区间,每两个数的间隔为一个区间
right	接收 boolean,代表右侧是否为闭区间,默认为 True
labels	接收 list、array,表示离散化后各个类别的名称,默认为空
retbins	接收 boolean,代表是否返回区间标签,默认为 False
precision	接收 int,显示标签的精度,默认为 3

【例 5-41】 cut 函数应用。

```
In[41]:  np.random.seed(666)
         score_list = np.random.randint(25, 100, size = 10)
         print('原始数据:\n',score_list)
         bins = [0, 59, 70, 80, 100]
         score_cut = pd.cut(score_list, bins)
         print(pd.value_counts(score_cut))
         # 统计每个区间人数
Out[41]: 原始数据:
         [27 70 55 87 95 98 55 61 86 76]
         (80, 100]  4
         (0, 59]    3
         (59, 70]   2
         (70, 80]   1
         dtype: int64
```

【例 5-42】 将泰坦尼克数据集中的年龄字段按下面规则分组转换为分类特征:(≤12,儿童)、(≤18,青少年)、(≤60,成人)、(>60,老人)

```
In[42]: import seaborn as sns
        import sys
        # 导入泰坦尼克数据集
        df = sns.load_dataset('titanic')
        display(df.head())
        df['ageGroup'] = pd.cut(
                                 df['age'],
                                 bins = [0, 13, 19, 61, sys.maxsize],
                                 labels = ['儿童', '青少年', '成人', '老人']
                                   )
        # sys.maxsize 是指可以存储的最大值
        display(df.head())
```

Out[42]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	ageGroup
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False	成人
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False	成人
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True	成人
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False	成人
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True	成人

使用等宽法离散化对数据分布具有较高要求,若数据分布不均匀,那么各个类的数目也会变得不均匀。

2. 等频法

cut 函数虽然不能够直接实现等频离散化,但可以通过定义将相同数量的记录放进每个区间。

【例 5-43】 等频法离散化连续型数据。

```
In[43]:  def SameRateCut(data,k):
             k = 2
             w = data.quantile(np.arange(0,1 + 1.0/k,1.0/k))
             data = pd.cut(data,w)
             return data
         result = SameRateCut(pd.Series(score_list),3)
         result.value_counts()
Out[43]: (73.0, 98.0]    5
         (27.0, 73.0]    4
         dtype: int64
```

相比较于等宽法,等频法避免了类分布不均匀的问题,但同时也有可能将数值非常接近的两个值分到不同的区间以满足每个区间对数据个数的要求。

3. 聚类分析法

一维聚类的方法包括两步,首先将连续型数据用聚类算法(如 K-Means 算法等)进行聚类,然后处理聚类得到的簇,为合并到一个簇的连续型数据做同一标记。聚类分析的离散化需要用户指定簇的个数来决定产生的区间数。

5.6 本章小结

本章主要针对数据预处理阶段的需求,介绍了使用 Pandas 数据载入、合并数据、数据清洗、数据标准化及数据转换的典型方法。

5.7 本章习题

一、单项选择题

1. 利用可视化绘图(　　)可以发现数据的异常点。

A. 密度图　　B. 直方图　　C. 盒图　　D. 概率图

2. 以下关于缺失值检测的说法中,正确的是(　　)。

A. null 和 notnull 可以对缺失值进行处理

B. dropna 方法既可以删除观测记录,也可以删除特征

C. fillna 方法中用来替换缺失值的值只能是数据框

D. Pandas 库中的 interpolate 模块包含多种插值方法

二、判断对错题

1. Pandas 中利用 merge 函数合并数据表时默认的是内连接方式。　　(　　)

2. Pandas 中的描述性统计一般会包括缺失数据。　　(　　)

3. 语句 dataframe.dropna(thresh=len(df)*0.9,axis=1) 表示如果某列的缺失值超过 90%则删除该列。　　(　　)

4. 利用 merge 方法合并数据时允许合并的 DataFrame 之间没有连接键。　　(　　)

5. 哑变量(Dummy Variables)又称虚拟变量,是用以反映质的属性的一个人工变量。　　(　　)

三、简答题

1. 简述 Pandas 删除空缺值方法 dropna 中参数 thresh 的使用方法。

2. 简述 Python 中利用数据统计方法检测异常值的常用方法及其原理。

3. 简述数据分析中要进行数据标准化的主要原因。

4. 简述 Pandas 中利用 cut 方法进行数据离散化的用法。

5.8 本章实训

本实训将第 4 章用到的小费数据集进行随机修改后(tips_mod.xls)进行预处理。

1. 导入模块

```
In[1]:   import pandas as pd
         import numpy as np
```

2. 获取数据

导入待处理数据 tips_mod.xls,并显示前 5 行。

```
In[2]:   fdata = pd.read_excel('D:/dataset/tips_mod.xls')
```

```
In[2]:  fdata.head()
```

Out[2]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

3. 分析数据

(1) 查看数据的描述信息。

```
In[3]:  print(fdata.shape)
        fdata.describe()
Out[3]:     (244.7)
```

	total_bill	tip	size
count	241.000000	241.000000	241.000000
mean	19.756141	2.997842	2.568465
std	8.933394	1.379711	0.951140
min	3.070000	1.000000	1.000000
25%	13.280000	2.000000	2.000000
50%	17.780000	2.920000	2.000000
75%	24.080000	3.550000	3.000000
max	50.810000	10.000000	6.000000

通过结果可以看出,共有 244 条记录,通过每个字段的均值和方差,看不出数据有异常。

(2) 显示聚餐时间段 time 的不重复值。

```
In[4]:  fdata['time'].unique()
Out[4]: array(['Dinner', 'Diner', 'Dier', 'Lunch', nan], dtype = object)
```

从结果发现有两个拼写错误“Diner”和“Dier”。

(3) 修改拼写错误的字段值。

```
In[5]:  fdata.ix[fdata['time'] == 'Diner','time'] = 'Dinner'
        fdata.ix[fdata['time'] == 'Dier','time'] = 'Dinner'
        fdata['time'].unique()
Out[5]: array(['Dinner', 'Lunch', nan], dtype = object)
```

(4) 检测数据中的缺失值。

```
In[6]:  fdata.isnull().sum()
Out[6]: total_bill      3
        tip             3
        sex             2
        smoker          0
```

```
day             0
time            2
size            3
dtype: int64
```

(5) 删除一行内有两个缺失值的数据。

```
In[7]:  fdata.dropna(thresh = 6,inplace = True)
        fdata.isnull().sum()
Out[7]: total_bill      2
        tip             3
        sex             2
        smoker          0
        day             0
        time            2
        size            3
        dtype: int64
```

(6) 删除 sex 或 time 为空的行。

```
In[8]:  fdata.dropna(subset = ['sex','time'],inplace = True)
        fdata.isnull().sum()
Out[8]: total_bill      2
        tip             3
        sex             0
        smoker          0
        day             0
        time            0
        size            3
        dtype: int64
```

(7) 对剩余有空缺的数据用平均值替换。

```
In[9]:  fdata.fillna(data.mean(),inplace = True)
        fdata.isnull().sum()
Out[9]: total_bill      0
        tip             0
        sex             0
        smoker          0
        day             0
        time            0
        size            0
        dtype: int64
```